

Tutorial: XML programming in Java

Doug Tidwell

Cyber Evangelist, developerWorks XML Team
September 1999

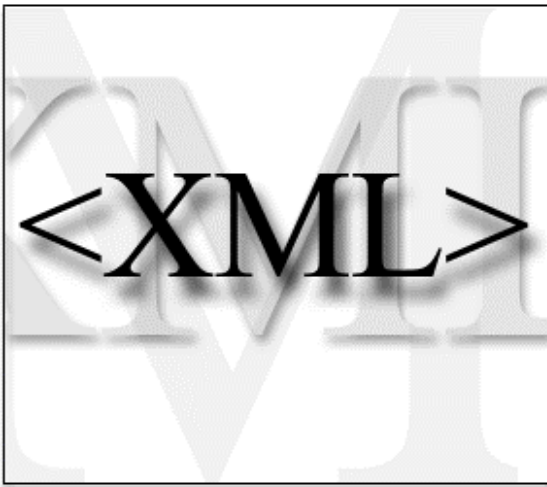
About this tutorial

Our first tutorial, "[Introduction to XML](#)," discussed the basics of XML and demonstrated its potential to revolutionize the Web. This tutorial shows you how to use an XML parser and other tools to create, process, and manipulate XML documents. Best of all, every tool discussed here is freely available at IBM's alphaWorks site (www.alphaworks.ibm.com) and other places on the Web.

About the author

Doug Tidwell is a Senior Programmer at IBM. He has well over a seventh of a century of programming experience and has been working with XML-like applications for several years. His job as a Cyber Evangelist is basically to look busy, and to help customers evaluate and implement XML technology. Using a specially designed pair of zircon-encrusted tweezers, he holds a Masters Degree in Computer Science from Vanderbilt University and a Bachelors Degree in English from the University of Georgia.

Section 1 – Introduction



About this tutorial

Our [previous tutorial](#) discussed the basics of XML and demonstrated its potential to revolutionize the Web. In this tutorial, we'll discuss how to use an XML parser to:

- Process an XML document
- Create an XML document
- Manipulate an XML document

We'll also talk about some useful, lesser-known features of XML parsers. Best of all, every tool discussed here is freely available at [IBM's alphaWorks site \(www.alphaworks.ibm.com\)](#) and other places on the Web.

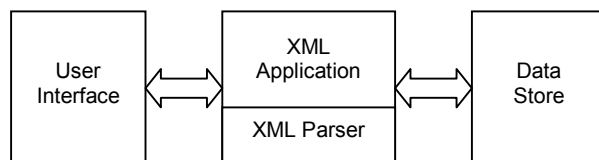


What's not here

There are several important programming topics *not* discussed here:

- Using visual tools to build XML applications
- Transforming an XML document from one vocabulary to another
- Creating interfaces for end users or other processes, and creating interfaces to back-end data stores

All of these topics are important when you're building an XML application. We're working on new tutorials that will give these subjects their due, so watch this space!



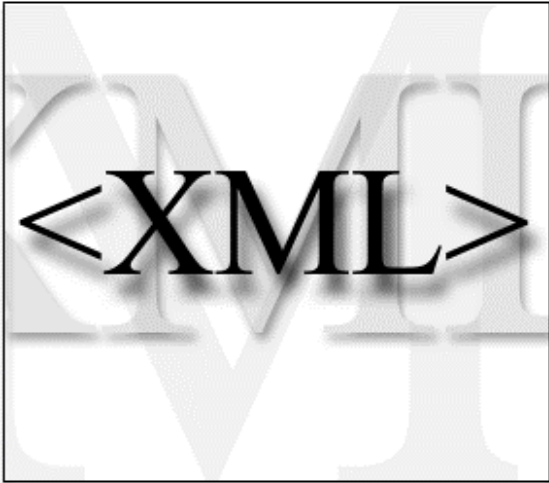
(Original artwork drawn by Doug Tidwell. All rights reserved.)

XML application architecture

An XML application is typically built around an XML parser. It has an interface to its users, and an interface to some sort of back-end data store.

This tutorial focuses on writing Java code that uses an XML parser to manipulate XML documents. In the beautiful picture on the left, this tutorial is focused on the middle box.

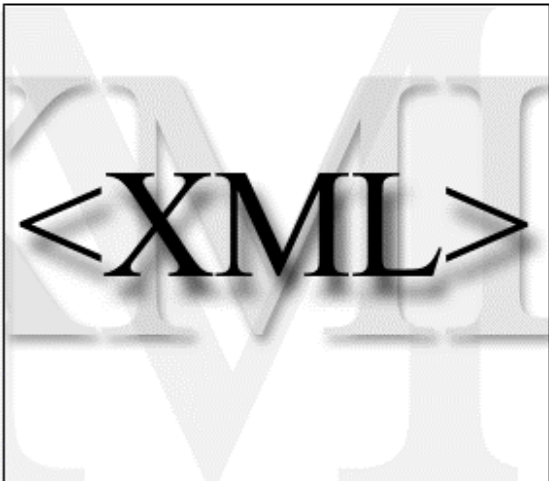
Section 2 – Parser basics



The basics

An XML parser is a piece of code that reads a document and analyzes its structure. In this section, we'll discuss how to use an XML parser to read an XML document. We'll also discuss the different types of parsers and when you might want to use them.

Later sections of the tutorial will discuss what you'll get back from the parser and how to use those results.

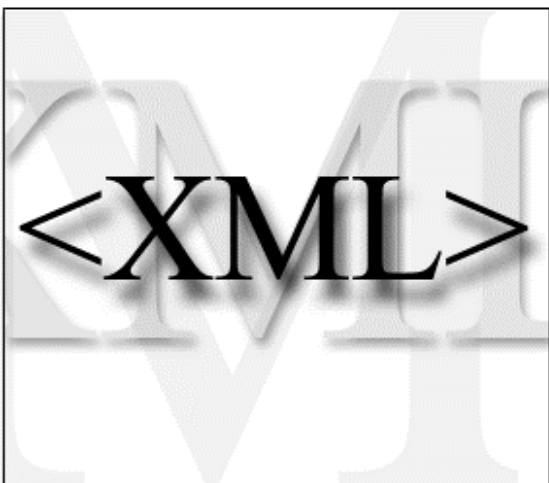


How to use a parser

We'll talk about this in more detail in the following sections, but in general, here's how you use a parser:

1. Create a parser object
2. Pass your XML document to the parser
3. Process the results

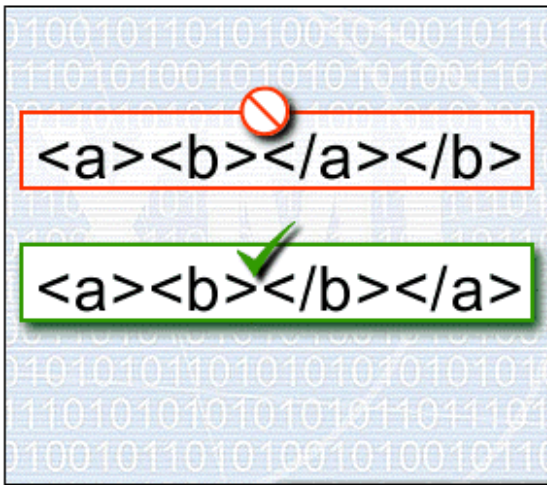
Building an XML application is obviously more involved than this, but this is the typical flow of an XML application.



Kinds of parsers

There are several different ways to categorize parsers:

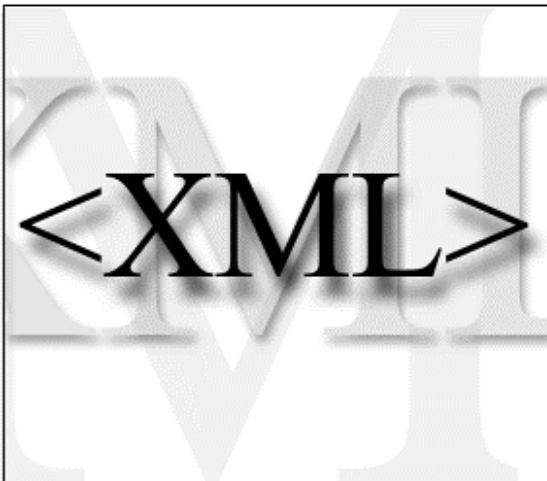
- Validating versus non-validating parsers
- Parsers that support the Document Object Model (DOM)
- Parsers that support the Simple API for XML (SAX)
- Parsers written in a particular language (Java, C++, Perl, etc.)



Validating versus non-validating parsers

As we mentioned in our first tutorial, XML documents that use a DTD and follow the rules defined in that DTD are called *valid documents*. XML documents that follow the basic tagging rules are called *well-formed documents*.

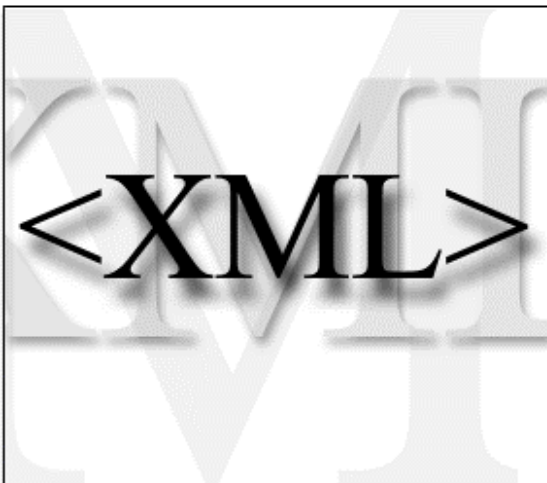
The XML specification requires all parsers to report errors when they find that a document is not well-formed. Validation, however, is a different issue. *Validating parsers* validate XML documents as they parse them. *Non-validating parsers* ignore any validation errors. In other words, if an XML document is well-formed, a non-validating parser doesn't care if the document follows the rules specified in its DTD (if any).



Why use a non-validating parser?

Speed and efficiency. It takes a significant amount of effort for an XML parser to process a DTD and make sure that every element in an XML document follows the rules of the DTD. If you're sure that an XML document is valid (maybe it was generated by a trusted source), there's no point in validating it again.

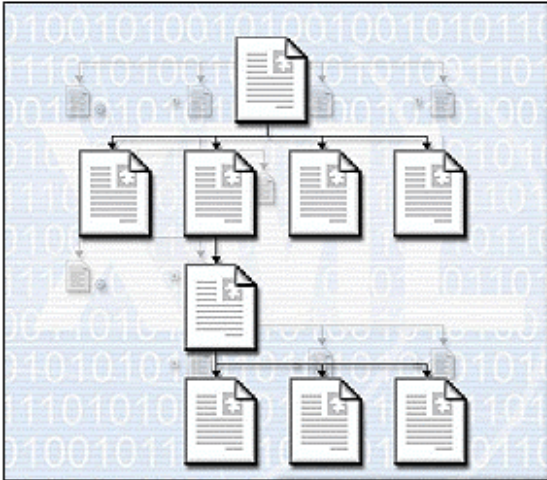
Also, there may be times when all you care about is finding the XML tags in a document. Once you have the tags, you can extract the data from them and process it in some way. If that's all you need to do, a non-validating parser is the right choice.



The Document Object Model (DOM)

The Document Object Model is an official recommendation of the World Wide Web Consortium (W3C). It defines an interface that enables programs to access and update the style, structure, and contents of XML documents. XML parsers that support the DOM implement that interface.

The first version of the specification, DOM Level 1, is available at <http://www.w3.org/TR/REC-DOM-Level-1>, if you enjoy reading that kind of thing.



What you get from a DOM parser

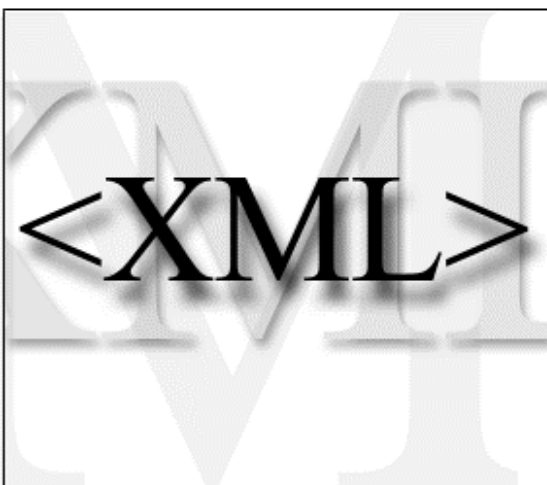
When you parse an XML document with a DOM parser, you get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.



A word about standards

Now that we're getting into developing XML applications, we might as well mention the XML specification. Officially, XML is a trademark of MIT and a product of the World Wide Web Consortium (W3C).

The [XML Specification](http://www.w3.org/TR/REC-xml), an official recommendation of the W3C, is available at www.w3.org/TR/REC-xml for your reading pleasure. The W3C site contains specifications for XML, DOM, and literally dozens of other XML-related standards. The XML zone at developerWorks has [an overview of these standards, complete with links to the actual specifications](#).



The Simple API for XML (SAX)

The SAX API is an alternate way of working with the contents of XML documents. A *de facto* standard, it was developed by David Megginson and other members of the XML-Dev mailing list.

To see the complete SAX standard, check out www.megginson.com/SAX/. To subscribe to the XML-Dev mailing list, send a message to majordomo@ic.ac.uk containing the following:
subscribe xml-dev.



What you get from a SAX parser

When you parse an XML document with a SAX parser, the parser generates events at various points in your document. It's up to you to decide what to do with each of those events.

A SAX parser generates events at the start and end of a document, at the start and end of an element, when it finds characters inside an element, and at several other points. You write the Java code that handles each event, and you decide what to do with the information you get from the parser.

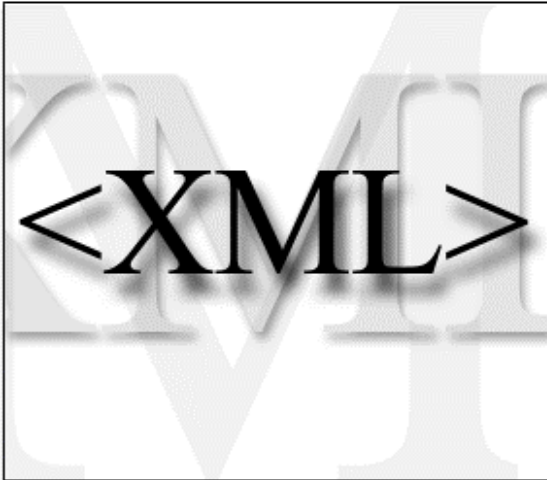
Why use SAX? Why use DOM?

We'll talk about this in more detail later, but in general, you should use a DOM parser when:

- You need to know a lot about the structure of a document
- You need to move parts of the document around (you might want to sort certain elements, for example)
- You need to use the information in the document more than once



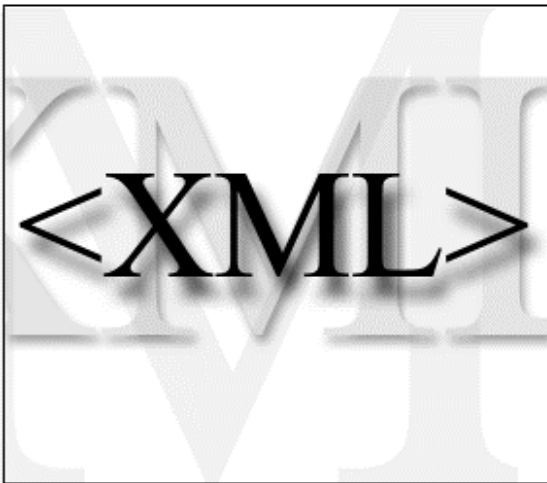
Use a SAX parser if you only need to extract a few elements from an XML document. SAX parsers are also appropriate if you don't have much memory to work with, or if you're only going to use the information in the document once (as opposed to parsing the information once, then using it many times later).



XML parsers in different languages

XML parsers and libraries exist for most languages used on the Web, including Java, C++, Perl, and Python. The next panel has links to XML parsers from IBM and other vendors.

Most of the examples in this tutorial deal with IBM's XML4J parser. All of the code we'll discuss in this tutorial uses standard interfaces. In the final section of this tutorial, though, we'll show you how easy it is to write code that uses another parser.



Resources – XML parsers

Java

- IBM's parser, XML4J, is available at www.alphaWorks.ibm.com/tech/xml4j.
- James Clark's parser, XP, is available at www.jclark.com/xml/xp.
- Sun's XML parser can be downloaded from developer.java.sun.com/developer/products/xml/ (you must be a member of the Java Developer Connection to download)
- DataChannel's XJParser is available at xdev.datachannel.com/downloads/xjparser/.

C++

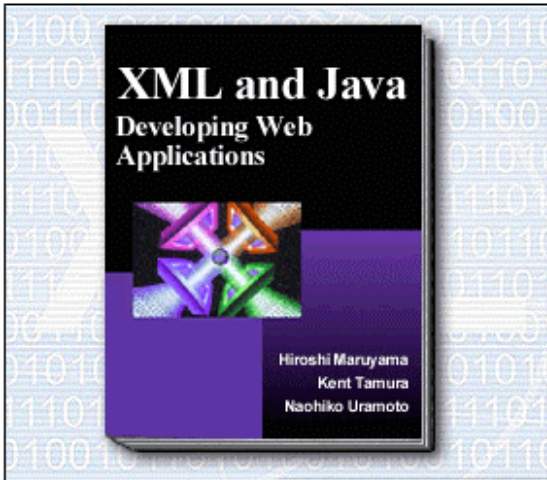
- IBM's XML4C parser is available at www.alphaWorks.ibm.com/tech/xml4c.
- James Clark's C++ parser, expat, is available at www.jclark.com/xml/expat.html.

Perl

- There are several XML parsers for Perl. For more information, see www.perlxml.com/faq/perl-xml-faq.html.

Python

- For information on parsing XML documents in Python, see www.python.org/topics/xml/.



One more thing

While we're talking about resources, there's one more thing: the best book on XML and Java (in our humble opinion, anyway).

We highly recommend [XML and Java: Developing Web Applications](#), written by Hiroshi Maruyama, Kent Tamura, and Naohiko Uramoto, the three original authors of IBM's XML4J parser. Published by Addison-Wesley, it's available at bookpool.com or your local bookseller.

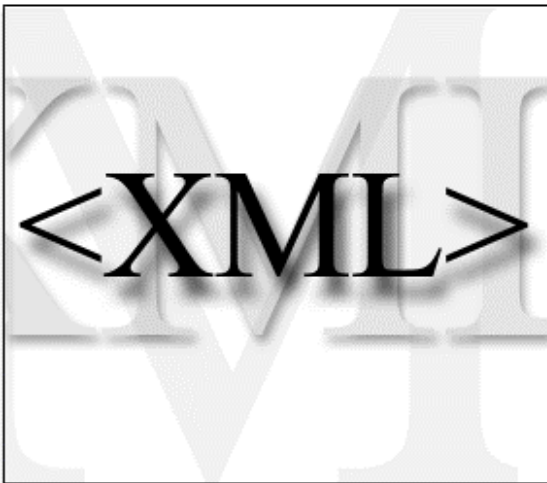
Summary

The heart of any XML application is an XML parser. To process an XML document, your application will create a parser object, pass it an XML document, then process the results that come back from the parser object.

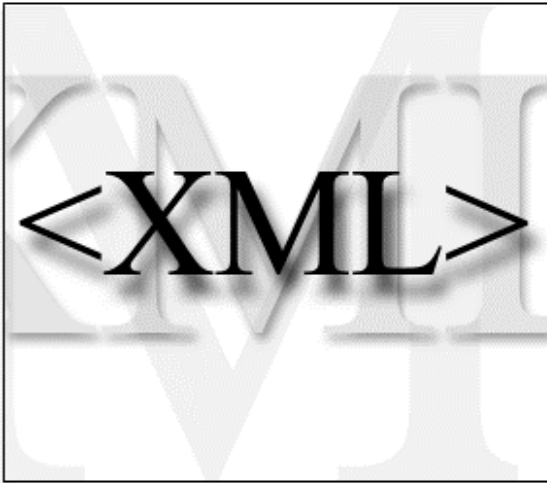
We've discussed the different kinds of XML parsers, and why you might want to use each one. We categorized parsers in several ways:

- Validating versus non-validating parsers
- Parsers that support the Document Object Model (DOM)
- Parsers that support the Simple API for XML (SAX)
- Parsers written in a particular language (Java, C++, Perl, etc.)

In our next section, we'll talk about DOM parsers and how to use them.



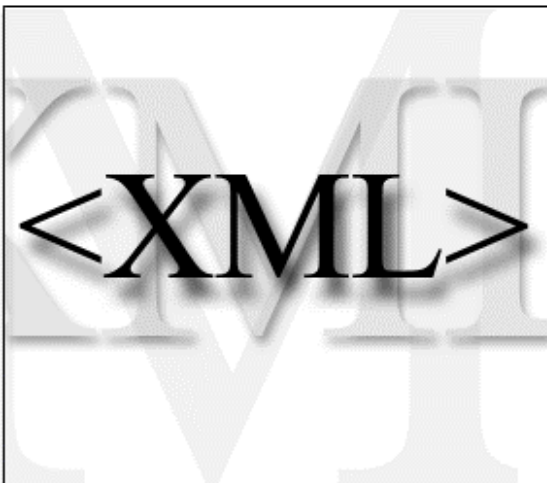
Section 3 – The Document Object Model (DOM)



♪ Dom, dom, dom, dom, dom, ♪
Doobie-doobie, ♪
♪ Dom, dom, dom, dom, dom...

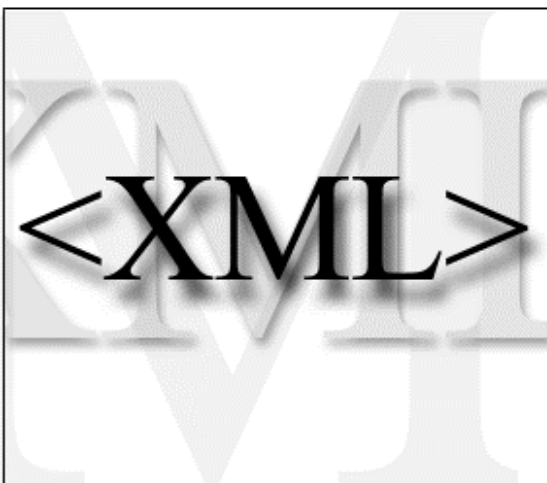
The DOM is a common interface for manipulating document structures. One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without changes. (We'll demonstrate this later.)

As we mentioned earlier, a DOM parser returns a tree structure that represents your entire document.



Sample code

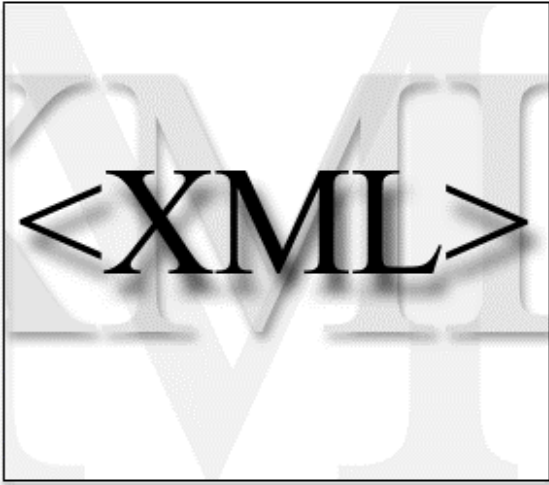
Before we go any further, make sure you've downloaded our sample XML applications onto your machine. Unzip the file [xmljava.zip](#), and you're ready to go! (Be sure to remember where you put the file.)



DOM interfaces

The DOM defines several Java interfaces. Here are the most common:

- `Node`: The base datatype of the DOM.
- `Element`: The vast majority of the objects you'll deal with are `Element`s.
- `Attr`: Represents an attribute of an element.
- `Text`: The actual content of an `Element` or `Attr`.
- `Document`: Represents the entire XML document. A `Document` object is often referred to as a *DOM tree*.



Common DOM methods

When you're working with the DOM, there are several methods you'll use often:

- `Document.getDocumentElement()`
Returns the root element of the document.
- `Node.getFirstChild()` and `Node.getLastChild()`
Returns the first or last child of a given `Node`.
- `Node.getNextSibling()` and `Node.getPreviousSibling()`
Deletes everything in the DOM tree, reformats your hard disk, and sends an obscene e-mail greeting to everyone in your address book. (*Not really.* These methods return the next or previous sibling of a given `Node`.)
- `Node.getAttribute(attrName)`
For a given `Node`, returns the attribute with the requested name. For example, if you want the `Attr` object for the attribute named `id`, use `getAttribute("id")`.

```
<?xml version="1.0"?>
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </author>
  <title>Sonnet 130</title>
  <line>My mistress' eyes are ...
```

Our first DOM application!

We've been at this a while, so let's go ahead and actually do something. Our first application simply reads an XML document and writes the document's contents to standard output.

At a command prompt, run this command:

```
java domOne sonnet.xml
```

This loads our application and tells it to parse the file `sonnet.xml`. If everything goes well, you'll see the contents of the XML document written out to standard output.

The `domOne.java` source code is on page 33.

```
public class domOne
{
    public void parseAndPrint(String uri)
    ...
    public void printDOMTree(Node node)
    ...
    public static void main(String argv[])
    ...
}
```

domOne to Watch Over Me

The source code for `domOne` is pretty straightforward. We create a new class called `domOne`; that class has two methods, `parseAndPrint` and `printDOMTree`.

In the main method, we process the command line, create a `domOne` object, and pass the file name to the `domOne` object. The `domOne` object creates a parser object, parses the document, then processes the DOM tree (*aka* the Document object) via the `printDOMTree` method.

We'll go over each of these steps in detail.

```
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: ... ");
        ...
        System.exit(1);
    }

    domOne d1 = new domOne();
    d1.parseAndPrint(argv[0]);
}
```

Process the command line

The code to process the command line is on the left. We check to see if the user entered anything on the command line. If not, we print a usage note and exit; otherwise, we assume the first thing on the command line (`argv[0]`, in Java syntax) is the name of the document. We ignore anything else the user might have entered on the command line.

We're using command line options here to simplify our examples. In most cases, an XML application would be built with servlets, Java Beans, and other types of components; and command line options wouldn't be an issue.

```
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: ... ");
        ...
        System.exit(1);
    }

    domOne d1 = new domOne();
    d1.parseAndPrint(argv[0]);
}
```

Create a domOne object

In our sample code, we create a separate class called `domOne`. To parse the file and print the results, we create a new instance of the `domOne` class, then tell our newly-created `domOne` object to parse and print the XML document.

Why do we do this? Because we want to use a recursive function to go through the DOM tree and print out the results. We can't do this easily in a static method such as `main`, so we created a separate class to handle it for us.

```
try
{
    DOMParser parser = new DOMParser();
    parser.parse(uri);
    doc = parser.getDocument();
}
```

Create a parser object

Now that we've asked our instance of `domOne` to parse and process our XML document, its first order of business is to create a new `Parser` object. In this case, we're using a `DOMParser` object, a Java class that implements the DOM interfaces. There are other parser objects in the XML4J package, such as `SAXParser`, `ValidatingSAXParser`, and `NonValidatingDOMParser`.

Notice that we put this code inside a `try` block. The parser throws an exception under a number of circumstances, including an invalid URI, a DTD that can't be found, or an XML document that isn't valid or well-formed. To handle this gracefully, we'll need to catch the exception.

```
try
{
    DOMParser parser = new DOMParser();
    parser.parse(uri);
    doc = parser.getDocument();
}
...
if (doc != null)
    printDOMTree(doc);
```

Parse the XML document

Parsing the document is done with a single line of code. When the parse is done, we get the `Document` object created by the parser.

If the `Document` object is not `null` (it will be `null` if something went wrong during parsing), we pass it to the `printDOMTree` method.

```
public void printDOMTree(Node node)
{
    int nodeType = Node.getNodeType();
    switch (nodeType)
    {
        case DOCUMENT_NODE:
            printDOMTree(((Document)node).
                GetDocumentElement());
            ...
        case ELEMENT_NODE:
            ...
            NodeList children =
                node.getChildNodes();
            if (children != null)
            {
                for (int i = 0;
                    i < children.getLength();
                    i++)
                    printDOMTree(children.item(i));
            }
    }
}
```

Process the DOM tree

Now that parsing is done, we'll go through the DOM tree. Notice that this code is recursive. For each node, we process the node itself, then we call the `printDOMTree` function recursively for each of the node's children. The recursive calls are shown at left.

Keep in mind that while some XML documents are very large, they don't tend to have many levels of tags. An XML document for the Manhattan phone book, for example, might have a million entries, but the tags probably wouldn't go more than a few layers deep. For this reason, stack overflow isn't a concern, as it is with other recursive algorithms.

```
Document Statistics for sonnet.xml:
=====
Document Nodes:          1
Element Nodes:          23
Entity Reference Nodes:  0
CDATA Sections:         0
Text Nodes:              45
Processing Instructions: 0
-----
Total:                   69 Nodes
```

Nodes a-plenty

If you look at `sonnet.xml`, there are twenty-four tags. You might think that would translate to twenty-four nodes. However, that's not the case. There are actually 69 nodes in `sonnet.xml`; one document node, 23 element nodes, and 45 text nodes. We ran `java domCounter sonnet.xml` to get the results shown on the left.

The `domCounter.java` source code is on page 35.

```
<?xml version="1.0"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
```

Sample node listing

For the fragment on the left, here are the nodes returned by the parser:

1. The Document node
2. The Element node corresponding to the `<sonnet>` tag
3. A Text node containing the carriage return at the end of the `<sonnet>` tag and the two spaces in front of the `<author>` tag
4. The Element node corresponding to the `<author>` tag
5. A Text node containing the carriage return at the end of the `<author>` tag and the four spaces in front of the `<last-name>` tag
6. The Element node corresponding to the `<last-name>` tag
7. A Text node containing the characters "Shakespeare"

If you look at all the blank spaces between tags, you can see why we get so many more nodes than you might expect.

```
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </author>
  <title>Sonnet 130</title>
  <line>
    <line>My mistress' eyes are nothing
like the sun,</line>
```

All those text nodes

If you go through a detailed listing of all the nodes returned by the parser, you'll find that a lot of them are pretty useless. All of the blank spaces at the start of the lines at the left are `Text` nodes that contain ignorable whitespace characters.

Notice that we wouldn't get these useless nodes if we had run all the tags together in a single line. We added the line breaks and spaces to our example to make it easier to read.

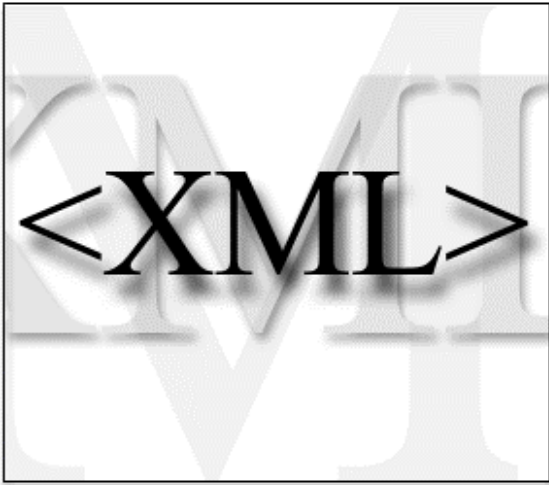
If human readability isn't necessary when you're building an XML document, leave out the line breaks and spaces. That makes your document smaller, and the machine processing your document doesn't have to build all those useless nodes.

```
switch (nodeType)
{
  case Node.DOCUMENT_NODE:
    ...
  case Node.ELEMENT_NODE:
    ...
  case Node.TEXT_NODE:
    ...
}
```

Know your Nodes

The final point we'll make is that in working with the `Nodes` in the DOM tree, we have to check the type of each `Node` before we work with it. Certain methods, such as `getAttributes`, return `null` for some node types. If you don't check the node type, you'll get unexpected results (at best) and exceptions (at worst).

The `switch` statement shown here is common in code that uses a DOM parser.



Summary

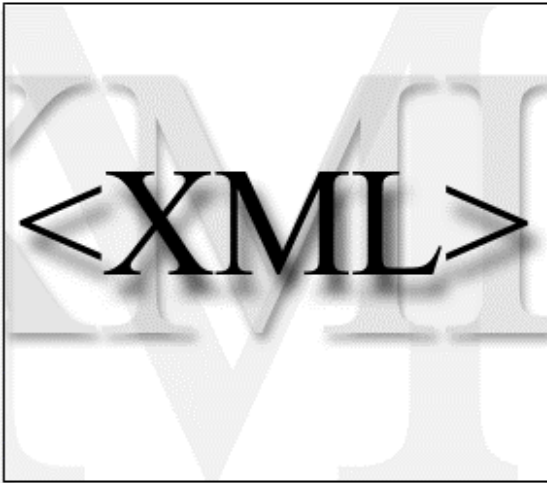
Believe it or not, that's about all you need to know to work with DOM objects. Our `domOne` code did several things:

- Created a `Parser` object
- Gave the `Parser` an XML document to parse
- Took the `Document` object from the `Parser` and examined it

In the final section of this tutorial, we'll discuss how to build a DOM tree without an XML source file, and show you how to sort elements in an XML document. Those topics build on the basics we've covered here.

Before we move on to those advanced topics, we'll take a closer look at the SAX API. We'll go through a set of examples similar to the ones in this section, illustrating the differences between SAX and DOM.

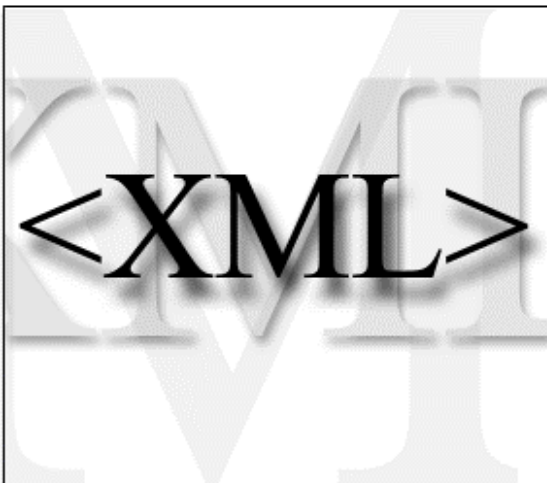
Section 4 – The Simple API for XML (SAX)



The Simple API for XML

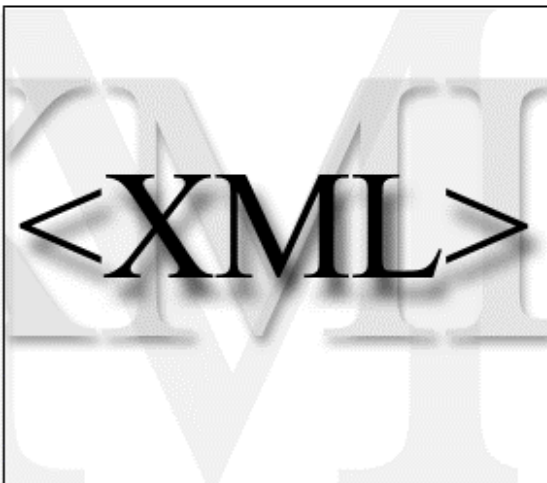
SAX is an event-driven API for parsing XML documents. In our DOM parsing examples, we sent the XML document to the parser, the parser processed the complete document, then we got a Document object representing our document.

In the SAX model, we send our XML document to the parser, and the parser notifies us when certain events happen. It's up to us to decide what we want to do with those events; if we ignore them, the information in the event is discarded.



Sample code

Before we go any further, make sure you've downloaded our sample XML applications onto your machine. Unzip the file [xmljava.zip](#), and you're ready to go! (Be sure to remember where you put the file.)

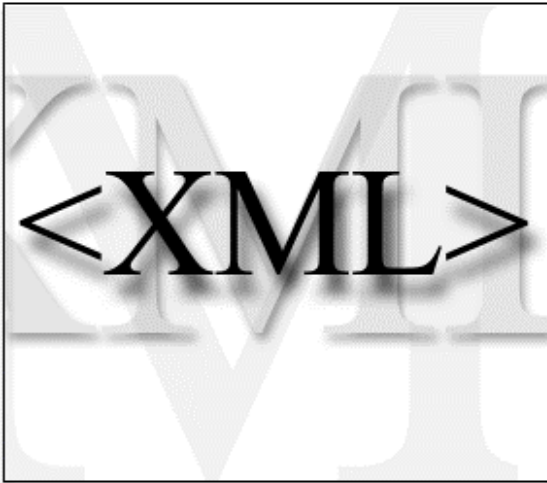


SAX events

The SAX API defines a number of events. You can write Java code that handles all of the events you care about. If you don't care about a certain type of event, you don't have to write any code at all. Just ignore the event, and the parser will discard it.

A wee listing of SAX events

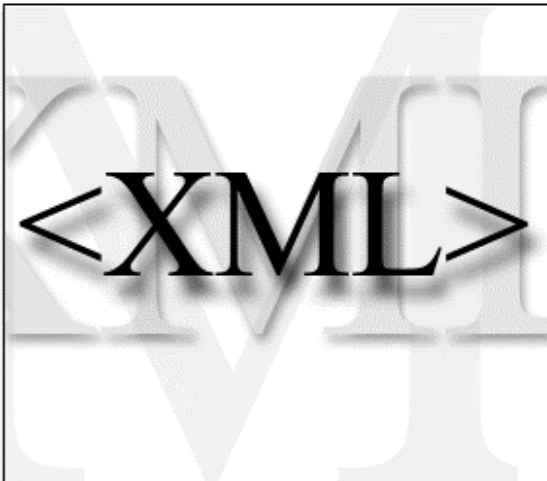
We'll list most of the SAX events here and on the next panel. All of the events on this panel are commonly used; the events on the next panel are more esoteric. They're part of the `HandlerBase` class in the `org.xml.sax` package.



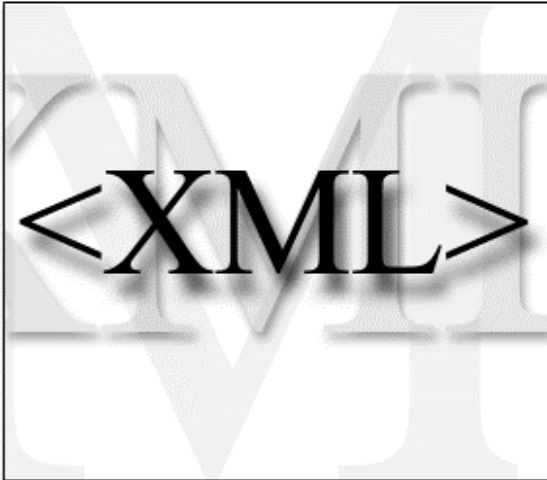
- `startDocument`
Signals the start of the document.
- `endDocument`
Signals the end of the document.
- `startElement`
Signals the start of an element. The parser fires this event when all of the contents of the opening tag have been processed. That includes the name of the tag and any attributes it might have.
- `endElement`
Signals the end of an element.
- `characters`
Contains character data, similar to a DOM Text node.

More SAX events

Here are some other SAX events:



- `ignorableWhitespace`
This event is analogous to the useless DOM nodes we discussed earlier. One benefit of this event is that it's different from the `character` event; if you don't care about whitespace, you can ignore all whitespace nodes by ignoring this event.
- `warning`, `error`, and `fatalError`
These three events indicate parsing errors. You can respond to them as you wish.
- `setDocumentLocator`
The parser sends you this event to allow you to store a SAX `Locator` object. The `Locator` object can be used to find out exactly where in the document an event occurred.



A note about SAX interfaces

The SAX API actually defines four interfaces for handling events: `EntityHandler`, `DTDHandler`, `DocumentHandler`, and `ErrorHandler`. All of these interfaces are implemented by `HandlerBase`.

Most of the time, your Java code will extend the `HandlerBase` class. If you want to subdivide the functions of your code (maybe you've got a great `DTDHandler` class already written), you can implement the `xxxHandler` classes individually.

```
<?xml version="1.0"?>
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </author>
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are ...
```

Our first SAX application!

Let's run our first SAX application. This application is similar to `domOne`, except it uses the SAX API instead of DOM.

At a command prompt, run this command:

```
java saxOne sonnet.xml
```

This loads our application and tells it to parse the file `sonnet.xml`. If everything goes well, you'll see the contents of the XML document written out to the console.

The `saxOne.java` source code is on page 37.

```
public class saxOne
  extends HandlerBase
  ...
  public void startDocument()
  ...
  public void
    startElement(String name,
                 AttributeList attrs)
  ...
  public void
    characters(char ch[], int start,
              int length)
```

saxOne overview

The structure of `saxOne` is different from `domOne` in several important ways. First of all, `saxOne` extends the `HandlerBase` class.

Secondly, `saxOne` has a number of methods, each of which corresponds to a particular SAX event. This simplifies our code because each type of event is completely handled by each method.

```

public void startDocument()
...
public void startElement(String name,
    AttributeList attrs)
...
public void characters(char ch[],
    int start, int length)
...
public void ignorableWhitespace(char ch[],
    int start, int length)
...
public void endElement(String name)
...
public void endDocument()
...
public void warning(SAXParseException ex)
...
public void error(SAXParseException ex)
...
public void fatalError(SAXParseException
    ex)
    throws SAXException
...

```

SAX method signatures

When you're extending the various SAX methods that handle SAX events, you need to use the correct method signature. Here are the signatures for the most common methods:

- `startDocument()` and `endDocument()`
These methods have no arguments.
- `startElement(String name, AttributeList attrs)`
`name` is the name of the element that just started, and `attrs` contains all of the element's attributes.
- `endElement(String name)`
`name` is the name of the element that just ended.
- `characters(char ch[], int start, int length)`
`ch` is an array of characters, `start` is the position in the array of the first character in this event, and `length` is the number of characters for this event.

```

public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: ...");
        ...
        System.exit(1);
    }

    saxOne s1 = new saxOne();
    s1.parseURI(argv[0]);
}

```

Process the command line

As in `domOne`, we check to see if the user entered anything on the command line. If not, we print a usage note and exit; otherwise, we assume the first thing on the command line is the name of the XML document. We ignore anything else the user might have entered on the command line.

```

public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: ...");
        ...
        System.exit(1);
    }

    saxOne s1 = new saxOne();
    s1.parseURI(argv[0]);
}

```

Create a saxOne object

In our sample code, we create a separate class called `saxOne`. The main procedure creates an instance of this class and uses it to parse our XML document. Because `saxOne` extends the `HandlerBase` class, we can use `saxOne` as an event handler for a SAX parser.

```
SAXParser parser = new SAXParser();
parser.setDocumentHandler(this);
parser.setErrorHandler(this);
```

```
try
{
    parser.parse(uri);
}
```

```
SAXParser parser = new SAXParser();
parser.setDocumentHandler(this);
parser.setErrorHandler(this);
```

```
try
{
    parser.parse(uri);
}
```

```
public void startDocument()
...
public void startElement(String name,
                        AttributeList attrs)
...
public void characters(char ch[],
                    int start, int length)
...
public void ignorableWhitespace(char ch[],
                            int start, int length)
...

```

Create a Parser object

Now that we've asked our instance of `saxOne` to parse and process our XML document, it first creates a new `Parser` object. In this sample, we use the `SAXParser` class instead of `DOMParser`.

Notice that we call two more methods, `setDocumentHandler` and `setErrorHandler`, before we attempt to parse our document. These functions tell our newly-created `SAXParser` to use `saxOne` to handle events.

Parse the XML document

Once our `SAXParser` object is set up, it takes a single line of code to process our document. As with `domOne`, we put the `parse` statement inside a `try` block so we can catch any errors that occur.

Process SAX events

As the `SAXParser` object parses our document, it calls our implementations of the SAX event handlers as the various SAX events occur. Because `saxOne` merely writes the XML document back out to the console, each event handler writes the appropriate information to `System.out`.

For `startElement` events, we write out the XML syntax of the original tag. For `character` events, we write the characters out to the screen. For `ignorableWhitespace` events, we write those characters out to the screen as well; this ensures that any line breaks or spaces in the original document will appear in the printed version.

Document Statistics for sonnet.xml:

```
=====
DocumentHandler Events:
  startDocument          1
  endDocument            1
  startElement           23
  endElement              23
  processingInstruction   0
  character               20
  ignorableWhitespace    25
ErrorHandler Events:
  warning                 0
  error                   0
  fatalError               0
-----
Total:                    93 Events
```

A cavalcade of ignorable events

As with the DOM, the SAX interface returns more events than you might think. We generated the listing at the left by running `java saxCounter sonnet.xml`.

One advantage of the SAX interface is that the 25 ignorableWhitespace events are simply ignored. We don't have to write code to handle those events, and we don't have to waste our time discarding them.

The `saxCounter.java` source code is on page 41.

```
<?xml version="1.0"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
```

Sample event listing

For the fragment on the left, here are the events returned by the parser:

1. A `startDocument` event
2. A `startElement` event for the `<sonnet>` element
3. An `ignorableWhitespace` event for the line break and the two blank spaces in front of the `<author>` tag
4. A `startElement` event for the `<author>` element
5. An `ignorableWhitespace` event for the line break and the four blank spaces in front of the `<last-name>` tag
6. A `startElement` event for the `<last-name>` tag
7. A `character` event for the characters "Shakespeare"
8. An `endElement` event for the `<last-name>` tag

```

...
<book id="1">
  <verse>
    Sing, O goddess, the anger of
    Achilles son of Peleus, that brought
    countless ills upon the Achaeans. Many
    a brave soul did it send hurrying down
    to Hades, and many a hero did it yield
    a prey to dogs and vultures, for so
    were the counsels of Jove fulfilled
    from the day on which the son of
    Atreus, king of men, and great
    Achilles, first fell out with one
    another.
  </verse>
  <verse>
    And which of the gods was it that set
    them on to quarrel? It was the son of
    Jove and Leto; for he was angry with
    the king and sent a pestilence upon
    ...

```

SAX versus DOM – part one

To illustrate the SAX API, we've taken our original `domOne` program and rewritten it to use SAX. To get an idea of the differences between the two, we'll talk about two parsing tasks.

For our first example, to parse *The Iliad* for all verses that contain the name "Agamemnon," the SAX API would be much more efficient. We would look for `startElement` events for the `<verse>` element, then look at each `character` event. We would save the character data from any event that contained the name "Agamemnon," and discard the rest.

Doing this with the DOM would require us to build Java objects to represent every part of the document, store those in a DOM tree, then search the DOM tree for `<verse>` elements that contained the desired text. This would take a lot of memory, and most of the objects created by the parser would be discarded without ever being used.

```

...
<address>
  <name>
    <title>Mrs.</title>
    <first-name>Mary</first-name>
    <last-name>McGoon</last-name>
  </name>
  <street>1401 Main Street</street>
  <city>Anytown</city>
  <state>NC</state>
  <zip>34829</zip>
</address>
<address>
  <name>
    ...

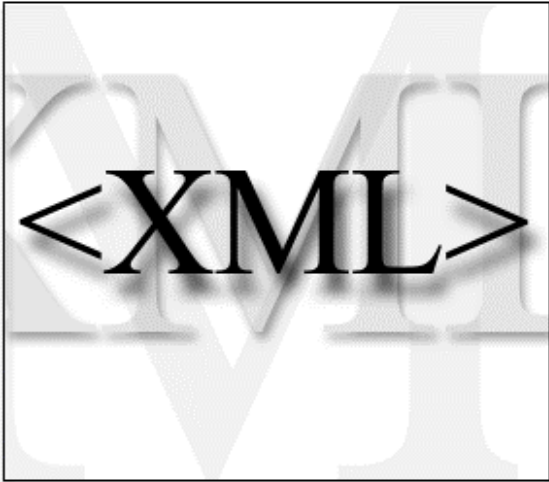
```

SAX versus DOM – part two

On the other hand, if we were parsing an XML document containing 10,000 addresses, and we wanted to sort them by last name, using the SAX API would make life very difficult for us.

We would have to build a data structure that stored every character and `startElement` event that occurred. Once we built all of these elements, we would have to sort them, then write a method that output the names in order.

Using the DOM API instead would save us a lot of time. DOM would automatically store all of the data, and we could use DOM functions to move the nodes in the DOM tree.

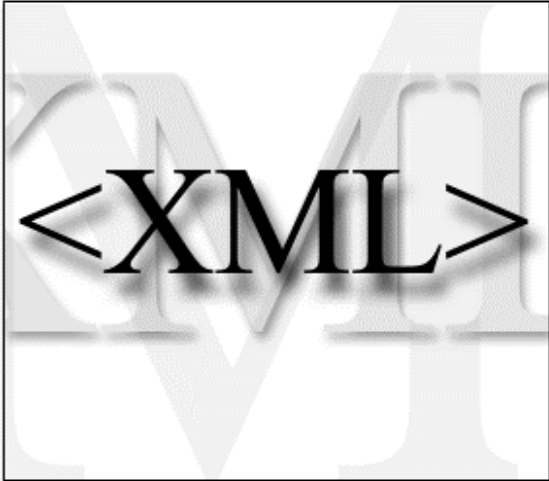


Summary

At this point, we've covered the two major APIs for working with XML documents. We've also discussed when you might want to use each one.

In our final topic, we'll discuss some advanced parser functions that you might need as you build an XML application.

Section 5 – Advanced parser functions



Overview

We've covered the basics of using an XML parser to process XML documents. In this section, we'll cover a couple of advanced topics.

First, we'll build a DOM tree from scratch. In other words, we'll create a `Document` object without using an XML source file.

Secondly, we'll show you how to use a parser to process an XML document contained in a string.

Next, we'll show you how to manipulate a DOM tree. We'll take our sample XML document and sort the lines of the sonnet.

Finally, we'll illustrate how using standard interfaces like DOM and SAX makes it easy to change parsers. We'll show you versions of two of our sample applications that use different XML parsers. None of the DOM and SAX code changes.

```
Document doc = (Document)Class.  
    forName("com.ibm.xml.dom.DocumentImpl").  
    newInstance();
```

Building a DOM tree from scratch

There may be times when you want to build a DOM tree from scratch. To do this, you create a `Document` object, then add various `Nodes` to it.

You can run `java domBuilder` to see an example application that builds a DOM tree from scratch. This application recreates the DOM tree built by the original parse of `sonnet.xml` (with the exception that it doesn't create whitespace nodes).

We begin by creating an instance of the `DocumentImpl` class. This class implements the `Document` interface defined in the DOM.

The `domBuilder.java` source code is on page 44.

```
Element root = doc.  
    createElement("sonnet");  
root.setAttribute("type",  
    "Shakespearean");
```

Adding Nodes to our Document

Now that we have our `Document` object, we can start creating `Nodes`. The first `Node` we'll create is a `<sonnet>` element. We'll create all the `Nodes` we need, then add each one to its appropriate parent.

Notice that we used the `setAttribute` method to set the value of the `type` attribute for the `<sonnet>` element.

```
Element author =  
    doc.createElement("author");  
  
Element lastName = doc.  
    createElement("last-name");  
lastName.appendChild(doc.  
    createTextNode("Shakespeare"));  
author.appendChild(lastName);
```

Establishing your document structure

As we continue to build our DOM tree, we'll need to create the structure of our document. To do this, we'll use the `appendChild` method appropriately. We'll create the `<author>` element, then create the various elements that belong beneath it, then use `appendChild` to add all of those elements to the correct parent.

Notice that `createElement` is a method of the `Document` class. Our `Document` object owns all of the elements we create here.

Finally, notice that we create `Text` nodes for the content of all elements. The `Text` node is the child of the element, and the `Text` node's parent is then added to the appropriate parent.

```
Element line14 = doc.  
    createElement("line");  
line14.appendChild(doc.  
    createTextNode("As any she ..."));  
text.appendChild(line14);  
root.appendChild(text);  
  
doc.appendChild(root);  
  
domBuilder db = new domBuilder();  
db.printDOMTree(doc);
```

Finishing our DOM tree

Once we've added everything to our `<sonnet>` element, we need to add it to the `Document` object. We call the `appendChild` method one last time, this time appending the child element to the `Document` object itself.

Remember that an XML document can have only one root element; `appendChild` will throw an exception if you try to add more than one root element to the `Document`.

When we have the DOM tree built, we create a `domBuilder` object, then call its `printDOMTree` method to print the DOM tree.



Using DOM objects to avoid parsing

You can think of a DOM `Document` object as the compiled form of an XML document. If you're using XML to move data from one place to another, you'll save a lot of time and effort if you can send and receive DOM objects instead of XML source.

This is one of the most common reasons why you might want to build a DOM tree from scratch.

In the worst case, you would have to create XML source from a DOM tree before you sent your data out, then you'd have to create a DOM tree when you received the XML data. Using DOM objects directly saves a great deal of time.

One caveat: be aware that a DOM object may be significantly larger than the XML source. If you have to send your data across a slow connection, sending the smaller XML source might more than make up for the wasted processing time spent reparsing your data.

Parsing an XML string

There may be times when you need to parse an XML string. IBM's XML4J parser supports this, although you have to convert your string into an `InputSource` object.

The first step is to create a `StringReader` object from your string. Once you've done that, you can create an `InputSource` from the `StringReader`.

You can run `java parseString` to see this code in action. In this sample application, the XML string is hardcoded; there are any number of ways you could get XML input from a user or another machine. With this technique, you don't have to write the XML document to a file system to parse it.

The `parseString.java` source code is on page 48.

```
parseString ps = new parseString();
StringReader sr =
    new StringReader("<?xml version=\"1.0\"?>
        <a>Alpha<b>Bravo</b>
        <c>Charlie</c></a>");
InputSource iSrc = new InputSource(sr);
ps.parseAndPrint(iSrc);
```

```

if (doc != null)
{
    sortLines(doc);
    printDOMTree(doc);
}
...
public void sortLines(Document doc)
{
    NodeList theLines =
        doc.getDocumentElement().
        getElementsByTagName("line");
    ...
}

```

Sorting Nodes in a DOM tree

To demonstrate how you can change the structure of a DOM tree, we'll change our DOM sample to sort the `<line>`s of the sonnet. There are several DOM methods that make it easy to move Nodes around the DOM tree.

To see this code in action, run `java domSorter sonnet.xml`. It doesn't do much for the rhyme scheme, but it does correctly sort the `<line>` elements.

To begin the task of sorting, we'll use the `getElementsByTagName` method to retrieve all of the `<line>` elements in the document. This method saves us the trouble of writing code to traverse the entire tree.

The `domSorter.java` source code is on page 50.

```

public String getTextFromLine(Node
                               lineElement)
{
    StringBuffer returnString =
        new StringBuffer();
    if (lineElement.getNodeName().
        equals("line"))
    {
        NodeList kids = lineElement.
            getChildNodes();
        if (kids != null)
            if (kids.item(0).getNodeName() ==
                Node.TEXT_NODE)
                returnString.append(kids.item(0).
                    getNodeValue());
    }
    else
        returnString.setLength(0);

    return new String(returnString);
}

```

Retrieving the text of our `<line>`s

To simplify the code, we created a helper function, `getTextFromLine`, that retrieves the text contained inside a `<line>` element. It simply looks at the `<line>` element's first child, and returns its text if that first child is a `Text` node.

This method returns a Java `String` so that our sort routine can use the `String.compareTo` method to determine the sorting order.

This code actually should check all of the `<line>`'s children, because it could contain entity references (say the entity `&miss;` was defined for the text "mistress"). We'll leave this improvement as an exercise for the reader.

```

public void sortLines(Document doc)
{
    NodeList theLines =
        doc.getDocumentElement().
            getElementsByTagName("line");
    if (theLines != null)
    {
        int len = theLines.getLength();
        for (int i=0; i < len; i++)
            for (int j=0; j < (len-1-i); j++)
                if (getTextFromLine(
                    theLines.item(j)).
                        compareTo(getTextFromLine(
                            theLines.item(j+1))) > 0)
                    theLines.item(j).
                        getParentNode().insertBefore(
                            theLines.item(j+1),
                            theLines.item(j));
    }
}

```

Sorting the text

Now that we have the ability to get the text from a given `<line>` element, we're ready to sort the data. Because we only have 14 elements, we'll use a bubble sort.

The bubble sort algorithm compares two adjacent values, and swaps them if they're out of order. To do the swap, we use the `getParentNode` and `insertBefore` methods.

`getParentNode` returns the parent of any `Node`; we use this method to get the parent of the current `<line>` (a `<lines>` element for documents using the sonnet DTD).

`insertBefore(nodeA, nodeB)` inserts `nodeA` into the DOM tree before `nodeB`. The most important feature of `insertBefore` is that if `nodeA` already exists in the DOM tree, it is removed from its current position and inserted before `nodeB`.

```

parentNode.appendChild(newChild);
...
parentNode.insertBefore(newChild);
...
parentNode.replaceChild(newChild,
                        oldChild);
...
parentNode.removeChild(oldChild)
...

```

Useful DOM methods for tree manipulation

In addition to `insertBefore`, there are several other DOM methods that are useful for tree manipulations.

- `parentNode.appendChild(newChild)`
Appends a node as the last child of a given parent node. Calling `parentNode.insertBefore(newChild, null)` does the same thing.
- `parentNode.replaceChild(newChild, oldChild)`
Replaces `oldChild` with `newChild`. The node `oldChild` must be a child of `parentNode`.
- `parentNode.removeChild(oldChild)`
Removes `oldChild` from `parentNode`.

```

/** Doesn't work */
for (Node kid = node.getFirstChild();
     kid != null;
     kid = kid.getNextSibling())
    node.removeChild(kid);

/** Does work */
while (node.hasChildNodes())
    node.removeChild(node.getFirstChild());

```

```

import com.sun.xml.parser.Parser;
import
    com.sun.xml.tree.XmlDocumentBuilder;

...

XmlDocumentBuilder builder =
    new XmlDocumentBuilder();
Parser parser =
    new com.sun.xml.parser.Parser();
parser.setDocumentHandler(builder);
builder.setParser(parser);
parser.parse(uri);
doc = builder.getDocument();

```

One more thing about tree manipulation

If you need to remove all the children of a given node, be aware that it's more difficult than it seems. Both code samples at the left look like they would work. However, only the one on the bottom actually works. The first sample doesn't work because `kid`'s instance data is updated as soon as `removeChild(kid)` is called.

In other words, the `for` loop removes `kid`, the first child, then checks to see if `kid.getNextSibling()` is null. Because `kid` has just been removed, it no longer has any siblings, so `kid.getNextSibling()` is null. The `for` loop will never run more than once. Whether `node` has one child or a thousand, the first code sample only removes the first child. Be sure to use the second code sample to remove all child nodes.

Using a different DOM parser

Although we can't think of a single reason why you'd want to, you *can* use a parser other than XML4J to parse your XML document. If you look at `domTwo.java`, you'll see that changing to Sun's XML parser required only two changes.

First of all, we had to import the files for Sun's classes. That's simple enough. The only other thing we had to change was the code that creates the `Parser` object. As you can see, setup for Sun's parser is a little more complicated, but the rest of the code is unchanged. All of the DOM code works without any changes.

Finally, the only other difference in `domTwo` is the command line format. For some reason, Sun's parser doesn't resolve file names in the same way. If you run `java domTwo file:///d:/sonnet.xml` (modifying the file URI based on your system, of course), you'll see the same results you saw with `domOne`.

The `domTwo.java` source code is on page 54.

```
import com.sun.xml.parser.Resolver;

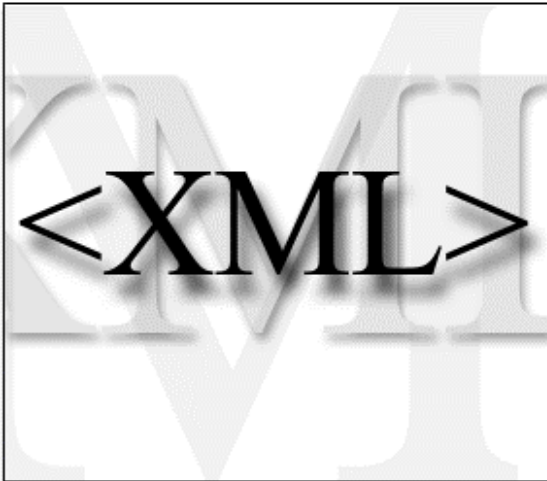
...
try
{
    Parser parser =
        ParserFactory.makeParser();
    parser.setDocumentHandler(this);
    parser.setErrorHandler(this);
    parser.parse(Resolver.
        createInputSource(new File(uri)));
}
}
```

Using a different SAX parser

We also created `saxTwo.java` to illustrate using Sun's SAX parser. As with `domTwo`, we made two basic changes. The first was to import Sun's `Resolver` class instead of IBM's `SAXParser` class.

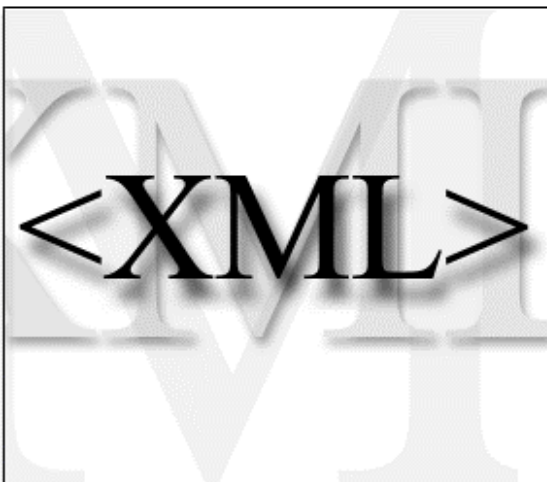
We had to change the line that creates the `Parser` object, and we had to create an `InputSource` object based on the URI we entered. The only other change we had to make is that the line that creates the parser has to be inside a `try` block in case we get an exception when we create the `Parser` object.

The `saxTwo.java` source code is on page 56.



Summary

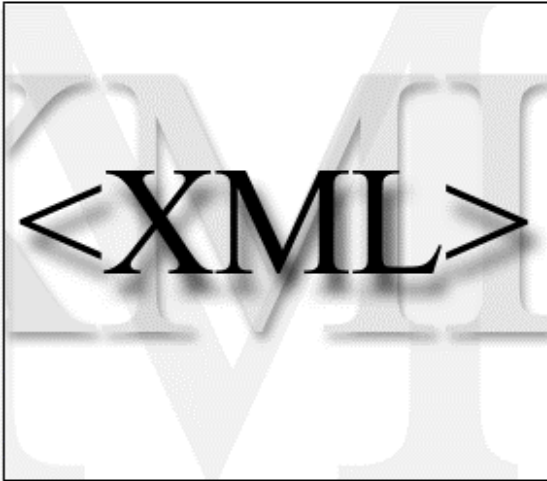
In this section, we've demonstrated some advanced coding techniques you can use with XML parsers. We demonstrated ways to generate DOM trees directly, how to parse strings as opposed to files, how to move items around in a DOM tree, and how changing parsers doesn't affect code written to the DOM and SAX standards.



Hope you enjoyed the show!

That's about it for this tutorial. We've talked about the basic architecture of XML applications, and we've shown you how to work with XML documents. Future tutorials will cover more details of building XML applications, including:

- Using visual tools to build XML applications
- Transforming an XML document from one vocabulary to another
- Creating front-end interfaces to end users or other processes, and creating back-end interfaces to data stores



For more information

If you'd like to know more about XML, check out [the XML zone of developerWorks](#). The site has code samples, other tutorials, information about XML standards efforts, and lots more.

Finally, we'd love to hear from you! We've designed developerWorks to be a resource for developers. If you have any comments, suggestions, or complaints about the site, let us know.

Thanks,
-[Doug Tidwell](#)

Appendix – Listings of our samples

This section lists all of the samples discussed in the tutorial. The listings include the Java source and the XML documents used as samples.

sonnet.xml

This is the sample XML document used throughout the tutorial.

```
<?xml version="1.0"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </author>
  <title>Sonnet 130</title>
  <text>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </text>
</sonnet>
```

sonnet.dtd

This is the DTD for our sample document.

```
<!-- sonnet.dtd -->
<!ELEMENT sonnet (author,title?,text) >
<!ATTLIST sonnet
      type (Shakespearean | Petrarchan) "Shakespearean">

<!ELEMENT text (line,line,line,line,
               line,line,line,line,
               line,line,line,line,
               line,line) >

<!ELEMENT author (last-name,first-name,nationality,
                 year-of-birth?,year-of-death?) >

<!ELEMENT title (#PCDATA)>
```

```

<!ELEMENT last-name (#PCDATA)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT year-of-birth (#PCDATA)>
<!ELEMENT year-of-death (#PCDATA)>
<!ELEMENT line (#PCDATA)>

```

domOne.java

This is our first DOM application. It parses an XML document and writes its contents to standard output.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import com.ibm.xml.parsers.*;

/**
 * domOne.java
 * Illustrates how to go through a DOM tree.
 */

public class domOne
{
    public void parseAndPrint(String uri)
    {
        Document doc = null;

        try
        {
            DOMParser parser = new DOMParser();
            parser.parse(uri);
            doc = parser.getDocument();
        }
        catch (Exception e)
        {
            System.err.println("Sorry, an error occurred: " + e);
        }

        // We've parsed the document now, so let's print it.

```

```

    if (doc != null)
        printDOMTree(doc);
}

/** Prints the specified node, then prints all of its children. */
public void printDOMTree(Node node)
{
    int type = node.getNodeType();
    switch (type)
    {
        // print the document element
        case Node.DOCUMENT_NODE:
        {
            System.out.println("<?xml version=\"1.0\" ?>");
            printDOMTree(((Document)node).getDocumentElement());
            break;
        }

        // print element with attributes
        case Node.ELEMENT_NODE:
        {
            System.out.print("<");
            System.out.print(node.getNodeName());
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
            {
                Node attr = attrs.item(i);
                System.out.print(" " + attr.getNodeName() +
                    "=\"" + attr.getNodeValue() +
                    "\"");
            }
            System.out.println(">");

            NodeList children = node.getChildNodes();
            if (children != null)
            {
                int len = children.getLength();
                for (int i = 0; i < len; i++)
                    printDOMTree(children.item(i));
            }

            break;
        }

        // handle entity reference nodes
        case Node.ENTITY_REFERENCE_NODE:
        {
            System.out.print("&");
            System.out.print(node.getNodeName());
            System.out.print(";");
            break;
        }

        // print cdata sections
        case Node.CDATA_SECTION_NODE:
        {
            System.out.print("<![CDATA[");
            System.out.print(node.getNodeValue());
            System.out.print("]]>");
            break;
        }

        // print text

```

```

    case Node.TEXT_NODE:
    {
        System.out.print(node.getNodeValue());
        break;
    }

    // print processing instruction
    case Node.PROCESSING_INSTRUCTION_NODE:
    {
        System.out.print("<?");
        System.out.print(node.getNodeName());
        String data = node.getNodeValue();
        {
            System.out.print(" ");
            System.out.print(data);
        }
        System.out.print(">");
        break;
    }
}

if (type == Node.ELEMENT_NODE)
{
    System.out.println();
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print('>');
}
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java domOne uri");
        System.out.println("  where uri is the URI of the XML document you want to
print.");
        System.out.println("  Sample: java domOne sonnet.xml");
        System.exit(1);
    }

    domOne d1 = new domOne();
    d1.parseAndPrint(argv[0]);
}
}

```

domCounter.java

This code parses an XML document, then goes through the DOM tree to gather statistics about the document. When the statistics are calculated, the code writes them to standard output.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result

```

```

* of using the Program. In no event will IBM be liable for any
* special, indirect or consequential damages or lost profits even if
* IBM has been advised of the possibility of their occurrence. IBM
* will not be liable for any third party claims against you.
*/

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import com.ibm.xml.parsers.DOMParser;

/**
 * domCounter.java
 * This code creates a DOM parser, parses a document, then
 * prints statistics about the number and type of nodes
 * found in the document.
 */

public class domCounter
{
    int documentNodes = 0;
    int elementNodes = 0;
    int entityReferenceNodes = 0;
    int cdataSections = 0;
    int textNodes = 0;
    int processingInstructions = 0;

    public void parseAndCount(String uri)
    {
        Document doc = null;

        try
        {
            DOMParser parser = new DOMParser();
            parser.parse(uri);
            doc = parser.getDocument();
        }
        catch (Exception e)
        {
            System.err.println("Sorry, an error occurred: " + e);
        }

        // We've parsed the document now, so let's scan the DOM tree and
        // print the statistics.

        if (doc != null)
        {
            scanDOMTree(doc);
            System.out.println("Document Statistics for " + uri + ":");
            System.out.println("=====");
            System.out.println("Document Nodes:           " + documentNodes);
            System.out.println("Element Nodes:           " + elementNodes);
            System.out.println("Entity Reference Nodes:   " + entityReferenceNodes);
            System.out.println("CDATA Sections:         " + cdataSections);
            System.out.println("Text Nodes:              " + textNodes);
            System.out.println("Processing Instructions:  " + processingInstructions);
            System.out.println("-----");
            int totalNodes = documentNodes + elementNodes + entityReferenceNodes +
                cdataSections + textNodes + processingInstructions;

```

```

        System.out.println("Total: " + totalNodes + " Nodes");
    }
}

/** Scans the DOM tree and counts the different types of nodes. */
public void scanDOMTree(Node node)
{
    int type = node.getNodeType();
    switch (type)
    {
        case Node.DOCUMENT_NODE:
            documentNodes++;
            scanDOMTree(((Document)node).getDocumentElement());
            break;

        case Node.ELEMENT_NODE:
            elementNodes++;
            NodeList children = node.getChildNodes();
            if (children != null)
            {
                int len = children.getLength();
                for (int i = 0; i < len; i++)
                    scanDOMTree(children.item(i));
            }
            break;

        case Node.ENTITY_REFERENCE_NODE:
            entityReferenceNodes++;
            break;

        case Node.CDATA_SECTION_NODE:
            cdataSections++;
            break;

        case Node.TEXT_NODE:
            textNodes++;
            break;

        case Node.PROCESSING_INSTRUCTION_NODE:
            processingInstructions++;
            break;
    }
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java domCounter uri");
        System.out.println("  where uri is the URI of your XML document.");
        System.out.println("  Sample: java domCounter sonnet.xml");
        System.exit(1);
    }

    domCounter dc = new domCounter();
    dc.parseAndCount(argv[0]);
}
}

```

saxOne.java

This is our first SAX application. It parses an XML document and writes its contents to standard output.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.xml.sax.AttributeList;
import org.xml.sax.HandlerBase;
import org.xml.sax.Parser;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.ParserFactory;

import com.ibm.xml.parsers.SAXParser;

/**
 * saxOne.java
 * This sample program illustrates how to use a SAX parser. It
 * parses a document and writes the document's contents back to
 * standard output.
 */

public class saxOne
    extends HandlerBase
{
    public void parseURI(String uri)
    {
        SAXParser parser = new SAXParser();
        parser.setDocumentHandler(this);
        parser.setErrorHandler(this);
        try
        {
            parser.parse(uri);
        }
        catch (Exception e)
        {
            System.err.println(e);
        }
    }

    /** Processing instruction. */
    public void processingInstruction(String target, String data)
    {
        System.out.print("<?");
        System.out.print(target);
        if (data != null && data.length() > 0)
        {
            System.out.print(' ');

```

```

        System.out.print(data);
    }
    System.out.print(">");

}

/** Start document. */
public void startDocument()
{
    System.out.println("<?xml version=\"1.0\"?>");
}

/** Start element. */
public void startElement(String name, AttributeList attrs)
{
    System.out.print("<");
    System.out.print(name);
    if (attrs != null)
    {
        int len = attrs.getLength();
        for (int i = 0; i < len; i++)
        {
            System.out.print(" ");
            System.out.print(attrs.getName(i));
            System.out.print("=\");
            System.out.print(attrs.getValue(i));
            System.out.print("\");
        }
    }
    System.out.print(">");
}

/** Characters. */
public void characters(char ch[], int start, int length)
{
    System.out.print(new String(ch, start, length));
}

/** Ignorable whitespace. */
public void ignorableWhitespace(char ch[], int start, int length)
{
    characters(ch, start, length);
}

/** End element. */
public void endElement(String name)
{
    System.out.print("</");
    System.out.print(name);
    System.out.print(">");
}

/** End document. */
public void endDocument()
{
    // No need to do anything.
}

//
// ErrorHandler methods
//

/** Warning. */

```

```

public void warning(SAXParseException ex)
{
    System.err.println("[Warning] "+
        getLocationString(ex)+" "+
        ex.getMessage());
}

/** Error. */
public void error(SAXParseException ex)
{
    System.err.println("[Error] "+
        getLocationString(ex)+" "+
        ex.getMessage());
}

/** Fatal error. */
public void fatalError(SAXParseException ex)
    throws SAXException
{
    System.err.println("[Fatal Error] "+
        getLocationString(ex)+" "+
        ex.getMessage());

    throw ex;
}

/** Returns a string of the location. */
private String getLocationString(SAXParseException ex)
{
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null)
    {
        int index = systemId.lastIndexOf('/');
        if (index != -1)
            systemId = systemId.substring(index + 1);
        str.append(systemId);
    }
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());

    return str.toString();
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java saxOne uri");
        System.out.println("  where uri is the URI of your XML document.");
        System.out.println("  Sample: java saxOne sonnet.xml");
        System.exit(1);
    }

    saxOne s1 = new saxOne();
    s1.parseURI(argv[0]);
}
}

```

saxCounter.java

This code parses an XML document and calculates statistics about the document as it receives SAX events. When the entire document has been parsed, the code writes the statistics to standard output.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.xml.sax.AttributeList;
import org.xml.sax.HandlerBase;
import org.xml.sax.Parser;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.ParserFactory;

import com.ibm.xml.parsers.SAXParser;

/**
 * saxCounter.java
 * This sample program calculates statistics for an XML document,
 * based on the SAX events received. When the parse is complete,
 * it prints the statistics to standard output.
 */

public class saxCounter
    extends HandlerBase
{
    int startDocumentEvents = 0;
    int endDocumentEvents = 0;
    int startElementEvents = 0;
    int endElementEvents = 0;
    int processingInstructionEvents = 0;
    int characterEvents = 0;
    int ignorableWhitespaceEvents = 0;
    int warningEvents = 0;
    int errorEvents = 0;
    int fatalErrorEvents = 0;

    public void parseURI(String uri)
    {
        SAXParser parser = new SAXParser();
        parser.setDocumentHandler(this);
        parser.setErrorHandler(this);
        try
        {

```

```

        parser.parse(uri);
    }
    catch (Exception e)
    {
        System.err.println(e);
    }

    System.out.println("Document Statistics for " + uri + ":");
    System.out.println("=====");
    System.out.println("DocumentHandler Events:");
    System.out.println("  startDocument      " +
        startDocumentEvents);
    System.out.println("  endDocument        " +
        endDocumentEvents);
    System.out.println("  startElement       " +
        startElementEvents);
    System.out.println("  endElement         " +
        endElementEvents);
    System.out.println("  processingInstruction " +
        processingInstructionEvents);
    System.out.println("  character          " +
        characterEvents);
    System.out.println("  ignorableWhitespace " +
        ignorableWhitespaceEvents);
    System.out.println("ErrorHandler Events:");
    System.out.println("  warning            " +
        warningEvents);
    System.out.println("  error              " +
        errorEvents);
    System.out.println("  fatalError         " +
        fatalErrorEvents);
    System.out.println("-----");
    int totalEvents = startDocumentEvents + endDocumentEvents +
        startElementEvents + endElementEvents +
        processingInstructionEvents +
        characterEvents + ignorableWhitespaceEvents +
        warningEvents + errorEvents + fatalErrorEvents;
    System.out.println("Total:      " +
        totalEvents + " Events");
}

/** Processing instruction. */
public void processingInstruction(String target, String data)
{
    processingInstructionEvents++;
}

/** Start document. */
public void startDocument()
{
    startDocumentEvents++;
}

/** Start element. */
public void startElement(String name, AttributeList attrs)
{
    startElementEvents++;
}

/** Characters. */
public void characters(char ch[], int start, int length)
{
    characterEvents++;
}

```

```
}

/** Ignorable whitespace. */
public void ignorableWhitespace(char ch[], int start, int length)
{
    ignorableWhitespaceEvents++;
}

/** End element. */
public void endElement(String name)
{
    endElementEvents++;
}

/** End document. */
public void endDocument()
{
    endDocumentEvents++;
}

//
// ErrorHandler methods
//

/** Warning. */
public void warning(SAXParseException ex)
{
    warningEvents++;
}

/** Error. */
public void error(SAXParseException ex)
{
    errorEvents++;
}

/** Fatal error. */
public void fatalError(SAXParseException ex)
    throws SAXException
{
    fatalErrorEvents++;
    throw ex;
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java saxCounter uri");
        System.out.println("    where uri is the URI of your XML document.");
        System.out.println("    Sample: java saxCounter sonnet.xml");
        System.exit(1);
    }

    saxCounter sc = new saxCounter();
    sc.parseURI(argv[0]);
}
}
```

domBuilder.java

This code builds a DOM tree without using an XML document as source. When the tree is complete, this code writes the tree's contents to standard output.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import com.ibm.xml.parsers.*;

/**
 * domBuilder.java
 * This sample program illustrates how to create a DOM tree from scratch.
 */

public class domBuilder
{
    /** Prints the specified node, recursively. */
    public void printDOMTree(Node node)
    {
        int type = node.getNodeType();
        switch (type)
        {
            // print the document element
            case Node.DOCUMENT_NODE:
            {
                System.out.println("<?xml version=\"1.0\" ?>");
                printDOMTree(((Document)node).getDocumentElement());
                break;
            }

            // print element with attributes
            case Node.ELEMENT_NODE:
            {
                System.out.print("<");
                System.out.print(node.getNodeName());
                NamedNodeMap attrs = node.getAttributes();
                for (int i = 0; i < attrs.getLength(); i++)
                {
                    Node attr = attrs.item(i);

```

```

        System.out.print(" " + attr.getNodeName() +
                        "=\"" + attr.getNodeValue() +
                        "\"");
    }
    System.out.println(">");

    NodeList children = node.getChildNodes();
    if (children != null)
    {
        int len = children.getLength();
        for (int i = 0; i < len; i++)
            printDOMTree(children.item(i));
    }

    break;
}

// handle entity reference nodes
case Node.ENTITY_REFERENCE_NODE:
{
    System.out.print("&");
    System.out.print(node.getNodeName());
    System.out.print(";");
    break;
}

// print cdata sections
case Node.CDATA_SECTION_NODE:
{
    System.out.print("<![CDATA[");
    System.out.print(node.getNodeValue());
    System.out.print("]]>");
    break;
}

// print text
case Node.TEXT_NODE:
{
    System.out.print(node.getNodeValue());
    break;
}

// print processing instruction
case Node.PROCESSING_INSTRUCTION_NODE:
{
    System.out.print("<?");
    System.out.print(node.getNodeName());
    String data = node.getNodeValue();
    {
        System.out.print(" ");
        System.out.print(data);
    }
    System.out.print(">");
    break;
}
}

if (type == Node.ELEMENT_NODE)
{
    System.out.println();
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print('>');
}

```

```

    }
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 1 && argv[0].equals("-help"))
    {
        System.out.println("Usage: java domBuilder");
        System.out.println("    This code builds a DOM tree, then prints it.");
        System.exit(1);
    }

    try
    {
        Document doc = (Document)Class.
            forName("com.ibm.xml.dom.DocumentImpl").
            newInstance();

        Element root = doc.createElement("sonnet");
        root.setAttribute("type", "Shakespearean");

        Element author = doc.createElement("author");

        Element lastName = doc.createElement("last-name");
        lastName.appendChild(doc.createTextNode("Shakespeare"));
        author.appendChild(lastName);

        Element firstName = doc.createElement("first-name");
        firstName.appendChild(doc.createTextNode("William"));
        author.appendChild(firstName);

        Element nationality = doc.createElement("nationality");
        nationality.appendChild(doc.createTextNode("British"));
        author.appendChild(nationality);

        Element yearOfBirth = doc.createElement("year-of-birth");
        yearOfBirth.appendChild(doc.createTextNode("1564"));
        author.appendChild(yearOfBirth);

        Element yearOfDeath = doc.createElement("year-of-death");
        yearOfDeath.appendChild(doc.createTextNode("1616"));
        author.appendChild(yearOfDeath);

        root.appendChild(author);

        Element title = doc.createElement("title");
        title.appendChild(doc.createTextNode("Sonnet 130"));
        root.appendChild(title);

        Element text = doc.createElement("text");

        Element line01 = doc.createElement("line");
        line01.appendChild(doc.createTextNode("My mistress' eyes are nothing like the
sun, "));
        text.appendChild(line01);

        Element line02 = doc.createElement("line");
        line02.appendChild(doc.createTextNode("Coral is far more red than her lips
red. "));
        text.appendChild(line02);

        Element line03 = doc.createElement("line");

```

```
    line03.appendChild(doc.createTextNode("If snow be white, why then her breasts  
are dun,"));  
    text.appendChild(line03);  
  
    Element line04 = doc.createElement("line");  
    line04.appendChild(doc.createTextNode("If hairs be wires, black wires grow on  
her head."));  
    text.appendChild(line04);  
  
    Element line05 = doc.createElement("line");  
    line05.appendChild(doc.createTextNode("I have seen roses damasked, red and  
white,"));  
    text.appendChild(line05);  
  
    Element line06 = doc.createElement("line");  
    line06.appendChild(doc.createTextNode("But no such roses see I in her  
cheeks."));  
    text.appendChild(line06);  
  
    Element line07 = doc.createElement("line");  
    line07.appendChild(doc.createTextNode("And in some perfumes is there more  
delight"));  
    text.appendChild(line07);  
  
    Element line08 = doc.createElement("line");  
    line08.appendChild(doc.createTextNode("Than in the breath that from my mistress  
reeks."));  
    text.appendChild(line08);  
  
    Element line09 = doc.createElement("line");  
    line09.appendChild(doc.createTextNode("I love to hear her speak, yet well I  
know"));  
    text.appendChild(line09);  
  
    Element line10 = doc.createElement("line");  
    line10.appendChild(doc.createTextNode("That music hath a far more pleasing  
sound."));  
    text.appendChild(line10);  
  
    Element line11 = doc.createElement("line");  
    line11.appendChild(doc.createTextNode("I grant I never saw a goddess go,"));  
    text.appendChild(line11);  
  
    Element line12 = doc.createElement("line");  
    line12.appendChild(doc.createTextNode("My mistress when she walks, treads on the  
ground."));  
    text.appendChild(line12);  
  
    Element line13 = doc.createElement("line");  
    line13.appendChild(doc.createTextNode("And yet, by Heaven, I think my love as  
rare"));  
    text.appendChild(line13);  
  
    Element line14 = doc.createElement("line");  
    line14.appendChild(doc.createTextNode("As any she belied with false compare."));  
    text.appendChild(line14);  
  
    root.appendChild(text);  
  
    doc.appendChild(root);  
  
    domBuilder db = new domBuilder();  
    db.printDOMTree(doc);
```

```

    }
    catch (Exception e)
    {
        System.err.println(e);
    }
}
}

```

parseString.java

This code illustrates how to parse a string that contains an XML document.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.io.Reader;
import java.io.StringReader;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import com.ibm.xml.parsers.*;

/**
 * parseString.java
 * This sample program illustrates how to parse an XML document
 * contained in a String.
 */

public class parseString
{
    public void parseAndPrint(InputSource xmlSource)
    {
        Document doc = null;

        try
        {
            DOMParser parser = new DOMParser();
            parser.parse(xmlSource);
            doc = parser.getDocument();
        }
        catch (Exception e)
        {
            System.err.println("Sorry, an error occurred: " + e);
        }
    }
}

```

```

    }

    // We've parsed the document now, so let's print it.

    if (doc != null)
        printDOMTree(doc);
}

/** Prints the specified node, recursively. */
public void printDOMTree(Node node)
{
    int type = node.getNodeType();
    switch (type)
    {
        // print the document element
        case Node.DOCUMENT_NODE:
        {
            System.out.println("<?xml version=\"1.0\" ?>");
            printDOMTree(((Document)node).getDocumentElement());
            break;
        }

        // print element with attributes
        case Node.ELEMENT_NODE:
        {
            System.out.print("<");
            System.out.print(node.getNodeName());
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
            {
                Node attr = attrs.item(i);
                System.out.print(" " + attr.getNodeName() +
                    "=\"" + attr.getNodeValue() +
                    "\"");
            }
            System.out.println(">");

            NodeList children = node.getChildNodes();
            if (children != null)
            {
                int len = children.getLength();
                for (int i = 0; i < len; i++)
                    printDOMTree(children.item(i));
            }

            break;
        }

        // handle entity reference nodes
        case Node.ENTITY_REFERENCE_NODE:
        {
            System.out.print("&");
            System.out.print(node.getNodeName());
            System.out.print(";");
            break;
        }

        // print cdata sections
        case Node.CDATA_SECTION_NODE:
        {
            System.out.print("<![CDATA[");
            System.out.print(node.getNodeValue());
            System.out.print("]]>");
        }
    }
}

```

```

        break;
    }

    // print text
    case Node.TEXT_NODE:
    {
        System.out.print(node.getNodeValue());
        break;
    }

    // print processing instruction
    case Node.PROCESSING_INSTRUCTION_NODE:
    {
        System.out.print("<?");
        System.out.print(node.getNodeName());
        String data = node.getNodeValue();
        {
            System.out.print(" ");
            System.out.print(data);
        }
        System.out.print(">");
        break;
    }
}

if (type == Node.ELEMENT_NODE)
{
    System.out.println();
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print('>');
}
}

/** Main program entry point. */
public static void main(String argv[])
{
    parseString ps = new parseString();
    StringReader sr = new StringReader("<?xml
version=\"1.0\"?><a>Alpha<b>Bravo</b><c>Charlie</c></a>");
    InputSource iSrc = new InputSource(sr);
    ps.parseAndPrint(iSrc);
}
}

```

domSorter.java

This code looks for all the <line> elements in the XML document, then sorts them. It illustrates how to manipulate a DOM tree.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if

```

```

* IBM has been advised of the possibility of their occurrence. IBM
* will not be liable for any third party claims against you.
*/

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import com.ibm.xml.parsers.*;

/**
 * domSorter.java
 * This sample program illustrates how to rearrange the nodes in a
 * DOM tree.
 */

public class domSorter
{
    public void parseAndSortLines(String uri)
    {
        Document doc = null;

        try
        {
            DOMParser parser = new DOMParser();
            parser.parse(uri);
            doc = parser.getDocument();
        }
        catch (Exception e)
        {
            System.err.println("Sorry, an error occurred: " + e);
        }

        // We've parsed the document now, so let's sort it and print it.

        if (doc != null)
        {
            sortLines(doc);
            printDOMTree(doc);
        }
    }

    public String getTextFromLine(Node lineElement)
    {
        StringBuffer returnString = new StringBuffer();

        if (lineElement.getNodeName().equals("line"))
        {
            NodeList kids = lineElement.getChildNodes();
            if (kids != null)
            {
                if (kids.item(0).getNodeType() == Node.TEXT_NODE)
                {
                    returnString.append(kids.item(0).getNodeValue());
                }
            }
        }
    }
}

```

```

        else
            returnString.setLength(0);

        return new String(returnString);
    }

    /** Sorts the <line> elements in the file.
     * It uses a bubble sort algorithm, since a
     * sonnet only has 14 lines. */
    public void sortLines(Document doc)
    {
        NodeList theLines = doc.getDocumentElement().
            getElementsByTagName("line");
        if (theLines != null)
        {
            int len = theLines.getLength();
            for (int i = 0; i < len; i++)
                for (int j = 0; j < (len - 1 - i); j++)
                    if (getTextFromLine(theLines.item(j)).
                        compareTo(getTextFromLine(theLines.item(j+1))) > 0)
                        theLines.item(j).getParentNode().
                            insertBefore(theLines.item(j+1),
                                theLines.item(j));
        }
    }

    /** Prints the specified node, recursively. */
    public void printDOMTree(Node node)
    {
        int type = node.getNodeType();
        switch (type)
        {
            // print the document element
            case Node.DOCUMENT_NODE:
            {
                System.out.println("<?xml version=\"1.0\" ?>");
                printDOMTree(((Document)node).getDocumentElement());
                break;
            }

            // print element with attributes
            case Node.ELEMENT_NODE:
            {
                System.out.print("<");
                System.out.print(node.getNodeName());
                NamedNodeMap attrs = node.getAttributes();
                for (int i = 0; i < attrs.getLength(); i++)
                {
                    Node attr = attrs.item(i);
                    System.out.print(" " + attr.getNodeName() +
                        "=\"" + attr.getNodeValue() +
                        "\"");
                }
                System.out.println(">");

                NodeList children = node.getChildNodes();
                if (children != null)
                {
                    int len = children.getLength();
                    for (int i = 0; i < len; i++)
                        printDOMTree(children.item(i));
                }
                break;
            }
        }
    }

```

```

    }

    // handle entity reference nodes
    case Node.ENTITY_REFERENCE_NODE:
    {
        System.out.print("&");
        System.out.print(node.getNodeName());
        System.out.print(";");
        break;
    }

    // print cdata sections
    case Node.CDATA_SECTION_NODE:
    {
        System.out.print("<![CDATA[");
        System.out.print(node.getNodeValue());
        System.out.print("]]>");
        break;
    }

    // print text
    case Node.TEXT_NODE:
    {
        if (node.getNodeValue().trim().length() > 0)
            System.out.print(node.getNodeValue());
        break;
    }

    // print processing instruction
    case Node.PROCESSING_INSTRUCTION_NODE:
    {
        System.out.print("<?");
        System.out.print(node.getNodeName());
        String data = node.getNodeValue();
        if (data != null && data.length() > 0)
        {
            System.out.print(" ");
            System.out.print(data);
        }
        System.out.print(">");
        break;
    }
}

if (type == Node.ELEMENT_NODE)
{
    System.out.println();
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print('>');
}
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java domSorter uri");
        System.out.println("    where uri is the URI of the XML document you want to
sort.");
        System.out.println("    Sample: java domSorter sonnet.xml");
        System.out.println();
    }
}

```

```

        System.out.println("    Note: Your XML document must use the sonnet DTD.");
        System.exit(1);
    }

    domSorter ds = new domSorter();
    ds.parseAndSortLines(argv[0]);
}
}

```

domTwo.java

This code is identical to `domOne.java`, except it uses Sun's XML parser instead of IBM's. It illustrates the portability of the DOM interfaces.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import com.sun.xml.parser.Parser;
import com.sun.xml.tree.XmlDocumentBuilder;

/**
 * domTwo.java
 * Illustrates how to go through a DOM tree. Identical to domOne,
 * except it uses Sun's XML parser instead of IBM's.
 */

public class domTwo
{
    public void parseAndPrint(String uri)
    {
        Document doc = null;

        try
        {
            XmlDocumentBuilder builder = new XmlDocumentBuilder();
            Parser parser = new com.sun.xml.parser.Parser();
            parser.setDocumentHandler(builder);
            builder.setParser(parser);
            builder.setDisableNamespaces(false);

```

```

    parser.parse(uri);
    doc = builder.getDocument();
}
catch (Exception e)
{
    System.err.println("Sorry, an error occurred: " + e);
}

// We've parsed the document now, so let's print it.

if (doc != null)
    printDOMTree(doc);
}

/** Prints the specified node, recursively. */
public void printDOMTree(Node node)
{
    int type = node.getNodeType();
    switch (type)
    {
        // print the document element
        case Node.DOCUMENT_NODE:
        {
            System.out.println("<?xml version=\"1.0\" ?>");
            printDOMTree(((Document)node).getDocumentElement());
            break;
        }

        // print element with attributes
        case Node.ELEMENT_NODE:
        {
            System.out.print("<");
            System.out.print(node.getNodeName());
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
            {
                Node attr = attrs.item(i);
                System.out.print(" " + attr.getNodeName() +
                    "=\"" + attr.getNodeValue() +
                    "\"");
            }
            System.out.println(">");

            NodeList children = node.getChildNodes();
            if (children != null)
            {
                int len = children.getLength();
                for (int i = 0; i < len; i++)
                    printDOMTree(children.item(i));
            }

            break;
        }

        // handle entity reference nodes
        case Node.ENTITY_REFERENCE_NODE:
        {
            System.out.print("&");
            System.out.print(node.getNodeName());
            System.out.print(";");
            break;
        }
    }
}

```

```

// print cdata sections
case Node.CDATA_SECTION_NODE:
{
    System.out.print("<![CDATA[");
    System.out.print(node.getNodeValue());
    System.out.print("]]>");
    break;
}

// print text
case Node.TEXT_NODE:
{
    System.out.print(node.getNodeValue());
    break;
}

// print processing instruction
case Node.PROCESSING_INSTRUCTION_NODE:
{
    System.out.print("<?");
    System.out.print(node.getNodeName());
    String data = node.getNodeValue();
    {
        System.out.print(" ");
        System.out.print(data);
    }
    System.out.print(">");
    break;
}
}

if (type == Node.ELEMENT_NODE)
{
    System.out.println();
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print(">");
}
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java domTwo uri");
        System.out.println("    where uri is the URI of the XML document you want to
print.");
        System.out.println("    Sample: java domTwo sonnet.xml");
        System.exit(1);
    }

    domTwo d2 = new domTwo();
    d2.parseAndPrint(argv[0]);
}
}

```

saxTwo.java

This code is identical to `saxOne.java`, except it uses Sun's XML parser *instead* of IBM's. It illustrates the portability of the SAX interfaces.

```

/*
 * (C) Copyright IBM Corp. 1999 All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantability and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

```

```

import java.io.File;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

```

```

import org.xml.sax.AttributeList;
import org.xml.sax.HandlerBase;
import org.xml.sax.Parser;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.ParserFactory;

```

```

import com.sun.xml.parser.Resolver;

```

```

/**
 * saxTwo.java
 * This sample program illustrates how to use a SAX parser. It
 * parses a document and writes the document's contents back to
 * standard output. It is identical to saxOne.java except that
 * it uses Sun's XML parser instead of IBM's.
 */

```

```

public class saxTwo
  extends HandlerBase
{
  public void parseURI(String uri)
  {
    try
    {
      Parser parser = ParserFactory.makeParser();
      parser.setDocumentHandler(this);
      parser.setErrorHandler(this);
      parser.parse(Resolver.createInputSource(new File(uri)));
    }
    catch (Exception e)
    {
      System.err.println(e);
    }
  }

  /** Processing instruction. */
  public void processingInstruction(String target, String data)
  {
    System.out.print("<?");
    System.out.print(target);
    if (data != null && data.length() > 0)
    {

```

```

        System.out.print(' ');
        System.out.print(data);
    }
    System.out.print(">");

}

/** Start document. */
public void startDocument()
{
    System.out.println("<?xml version=\"1.0\"?>");
}

/** Start element. */
public void startElement(String name, AttributeList attrs)
{
    System.out.print("<");
    System.out.print(name);
    if (attrs != null)
    {
        int len = attrs.getLength();
        for (int i = 0; i < len; i++)
        {
            System.out.print(" ");
            System.out.print(attrs.getName(i));
            System.out.print("=\");
            System.out.print(attrs.getValue(i));
            System.out.print("\");
        }
    }
    System.out.print(">");
}

/** Characters. */
public void characters(char ch[], int start, int length)
{
    System.out.print(new String(ch, start, length));
}

/** Ignorable whitespace. */
public void ignorableWhitespace(char ch[], int start, int length)
{
    characters(ch, start, length);
}

/** End element. */
public void endElement(String name)
{
    System.out.print("</");
    System.out.print(name);
    System.out.print(">");
}

/** End document. */
public void endDocument()
{
    // No need to do anything.
}

//
// ErrorHandler methods
//

```

```

/** Warning. */
public void warning(SAXParseException ex)
{
    System.err.println("[Warning] "+
                       getLocationString(ex)+" "+
                       ex.getMessage());
}

/** Error. */
public void error(SAXParseException ex)
{
    System.err.println("[Error] "+
                       getLocationString(ex)+" "+
                       ex.getMessage());
}

/** Fatal error. */
public void fatalError(SAXParseException ex)
    throws SAXException
{
    System.err.println("[Fatal Error] "+
                       getLocationString(ex)+" "+
                       ex.getMessage());
    throw ex;
}

/** Returns a string of the location. */
private String getLocationString(SAXParseException ex)
{
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null)
    {
        int index = systemId.lastIndexOf('/');
        if (index != -1)
            systemId = systemId.substring(index + 1);
        str.append(systemId);
    }
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());

    return str.toString();
}

/** Main program entry point. */
public static void main(String argv[])
{
    if (argv.length == 0)
    {
        System.out.println("Usage: java saxTwo uri");
        System.out.println("  where uri is the URI of your XML document.");
        System.out.println("  Sample: java saxTwo sonnet.xml");
        System.exit(1);
    }

    saxTwo s2 = new saxTwo();
    s2.parseURI(argv[0]);
}
}

```